C212/A592 Lab 5

Intro to Software Systems

Instructions:

- Review the requirements given below and Complete your work. Please compress all files (including main to test your work) into a zip file and submit it through Canvas. Do not include any binary files.
- The grading scheme is provided on Canvas
- Ask questions during the Lab hours to understand more.

Objectives: (100 points: 15+25+60)

What you will learn:

- Array (One Dimensional and 2 Dimensional) and ArrayList
- Writing class
- Unit Testing

Part 1: Longest run

Write a java method (with appropriate method name, return type and argument lists) and accompanying main function to test it. Your program generates a sequence of 20 random die tosses in an array and prints the die values, marking only the longest run, like this:

1 2 5 5 3 1 2 4 3 (2 2 2 2) 3 6 5 5 6 3 1

If there is more than one run of maximum length, mark the first one. Use java Random class and don't hardcode the sequence of numbers. Make the best decision about how the code should be split between two methods. Write Junit tests.

Part 2: Bulgarian Solitaire

You should write a java method (with appropriate method name, return type and argument lists) and accompanying main function to test this program.

In this exercise, you will model the game of *Bulgarian Solitaire* **using an ArrayList**. The game starts with 45 cards. (They need not be playing cards. Unmarked index cards work just as well.) Randomly divide them into some number of piles of random size. For example, you might start with piles of size 20, 5, 1, 9, and 10. In each round, you take one card from each pile, forming a new pile with these cards. For example, the sample starting configuration would be transformed into piles of size 19, 4, 8, 9, and 5. The solitaire is over when the piles have size 1, 2, 3, 4, 5, 6, 7, 8, and 9, in some order. (It can be shown that you always end up with such a configuration.)

In your program, produce a random starting configuration and print it. Then keep applying the solitaire step and print the result. Stop when the solitaire final configuration is reached.

Part 3: Matrix Library

We will be creating a class which will be given a 2-D array when instantiated, and then implement various methods to perform operations on the 2-D array.

Create the class with the following API. Then write main program to test all methods of your class. Please make sure your main method to test Matrix Library is in another class file and not in the Matrix class file!

```
public class Matrix {

        private boolean[][] matrix;
        private final int NUMROW;
        private final int NUMCOL;

        public Matrix(boolean[][] m) { }
        public String toString() { }

        public Matrix transposeMatrix() { }
        public boolean isSquareMatrix() { }
        public Matrix rotateClockwise() { }
        public Matrix rotateCounterClockwise() { }
        public String percentageTrue() { }
        public boolean isEqual (Matrix m) { }
}
```

1. *public Matrix(boolean[][] m) { }*
    a. Initialize the NUMROW and NUMCOL variables. These are the row and column dimensions of *m.*
    b. The constructor should initialize the *boolean[][] matrix*.
    c. Do not assume the input to the constructor is a NxN matrix
    The constructor and other methods need to work on matrices that have any column and row sizes. Methods should work on 4x4s, 4x2s, 7x5s, etc.

2. *public String toString() { }*
    a. returns a string value of the entire matrix. **To receive full points, string must be in this form**: Each value should have a comma and a space except the last one. There should be no space before the closing bracket. True is 1, false is 0.

    4x4 matrix
    [1, 1, 1, 1
     0, 1, 0, 1
     0, 1, 1, 1
     1, 1, 0, 1]

2x4 matrix
[0, 1, 1, 1
1, 1, 0, 1]

3. *public Matrix transposeMatrix() {}*
    a. Returns the transpose of the instance field *matrix* as a new Matrix object
    b. Include these two matrices below in your test cases for this method.

The **transpose** of a matrix is a new matrix whose rows are the columns of the original. (This makes the columns of the new matrix the rows of the original).

Here is a matrix and its transpose:

$$\begin{pmatrix} 5 & 4 & 3 \\ 4 & 0 & 4 \\ 7 & 10 & 3 \end{pmatrix}^{T} = \begin{pmatrix} 5 & 4 & 7 \\ 4 & 0 & 10 \\ 3 & 4 & 3 \end{pmatrix}$$

Another way to look at the transpose is that the element at row r column c in the original is placed at row c column r of the transpose. The element $\mathbf{a_{rc}}$ of the original matrix becomes element $\mathbf{a_{cr}}$ in the transposed matrix.

$$\begin{pmatrix} 5 & 4 \\ 4 & 0 \\ 7 & 10 \\ -1 & 8 \end{pmatrix}^{T}_{4x2} = \begin{pmatrix} 5 & 4 & 7 & -1 \\ 4 & 0 & 10 & 8 \end{pmatrix}_{2x4}$$

4x4 matrix
[1, 1, 1, 1
0, 1, 0, 1
0, 1, 1, 1
1, 1, 0, 1]

It should return the following as output:

4x4 matrix
[1, 0, 0, 1
1, 1, 1, 1
1, 0, 1, 0
1, 1, 1, 1]

4. *public boolean isSquareMatrix() { }*
    a. returns true if the instance field *matrix* is a square matrix i.e., NUMROW is equal to NUMCOL; otherwise returns false

5. *public Matrix rotateClockwise() { }*
    a. Returns the instance field *matrix* rotated 90° to the right

Example: [1, 1, 1, 1 → [1, 1, 1
1, 1, 0, 0 1, 1, 1
1, 1, 1, 1] 1, 0, 1
1, 0, 1]

6. *public Matrix rotateCounterClockwise() { }*
    a. Returns the instance field *matrix* rotated 90° to the left.

Example: [1, 1, 1 [1, 1, 1, 1
1, 1, 1 → 1, 1, 0, 0
1, 0, 1 1, 1, 1, 1]
1, 0, 1]

7. public String percentageTrue() { }
    a. Returns the percentage of indices that are true in the instance field *matrix*.
    b. b. For simplicity, always round up to the nearest whole percentage.

Example: [1, 1, 1, 1
0, 0, 0, 0 → "50%"
0, 0, 0, 0
1, 1, 1, 1]

8. *boolean isEqual (Matrix m)*
    a. returns true if the instance field *matrix* is equal to given matrix m. The matrices will be equal if they have the same number of rows and columns and every value is also equal; otherwise return false

9. As always, use the *main* as a test client to test each function. Write **JUnit tests** for all methods.

    Note: where line breaks are added in your toString() method, they will need to be added in Junit tests as well. For example, below is a unit test of calling toString() on a 3x3 matrix with all false indices:

```
assertEquals("[ 0, 0, 0\n
                0, 0, 0\n
                0, 0, 0]", myMatrix.toString());
```